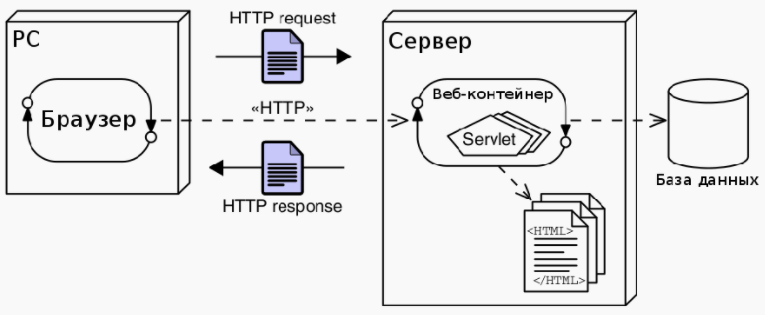
**https://jsehelper.blogspot.com/2016/01/jsp-servlets-jstl.html**

**Java-сервлеты. Особенности реализации, ключевые методы, преимущества и недостатки относительно CGI и FastCGI.**

* Сервлеты - это серверные сценарии,написанные на Java.
* Жизненным циклом сервлетов управляет веб-контейнер (он же контейнер сервлетов).
  + Веб сервер, это нечто обрабатывающее запросы с протоколом http
  + Веб контейнер (ака сервлет контейнер) - часть веб сервера обрабатывающее запросы согласно спецификации Java Servlet/JSP - программа, представляющая собой сервер, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл в соответствии с правилами, определёнными в спецификациях.
* В отличие от CGI, запросы обрабатываются в отдельных потоках (а не процессах) на веб-контейнере.



Преимущества и недостатки

* Преимущества сервлетов:
  + Выполняются быстрее, чем CGI-сценарии.
  + Хорошая масштабируемость.
  + Надёжность и безопасность (реализованы на Java).
  + Платформенно-независимы.
  + Множество инструментов мониторинга и отладки.
* Недостатки сервлетов:
  + Слабое разделение уровня представления и бизнес-логики.
  + Возможны конфликты при параллельной обработке запросов

Сервлеты

Сервлет - это класс, который расширяет функциональность класса HttpServlet и запускается внутри контейнера сервлетов.

Сервлет размещается на сервере, однако чтобы сервер мог использовать сервлет для обработки запросов, сервер должен поддерживать контейнер сервлетов (servlet container/engine). Например, Wildfly по сути является контейнером сервлетов.

Основные методы

Все методы в качестве параметра принимают два объекта: HttpServletRequest - хранит информацию о запросе и HttpServletResponse - управляет ответом на запрос.

* doGet: обрабатывает запросы GET (получение данных)
* doPost: обрабатывает запросы POST (отправка данных)
* doPut: обрабатывает запросы PUT (отправка данных для изменения)
* doDelete: обрабатывает запросы DELETE (удаление данных)
* doHead: обрабатывает запросы HEAD

Методы с вебконтейнером

* init() – Данный метод вызывается только один раз при создании сервлета. Обычно, создание происходит в момент перехода по ссылке (URL) на страницу, работу которой обеспечивает определённый сервлет. Это может быть изменено – и создание может происходить при запуске сервера.
* service() – При получении запроса, сервер создаёт новый поток, который вызывает метод service. Данный метод проверяет тип запроса (POST, GET и т.д.) и вызывает соответствующий метод (doPost, doGet и т.д.)
* destroy - Данный метод вызывается после того, как сервлет выполнил свои задачи. Используется, например, для таких задач, как запись данных в cookie, закрытие соединения к базе данных (далее – БД) и т.д.

**Контейнеры сервлетов. Жизненный цикл сервлета.**

Контейнер сервлетов – Это программа, которая запускается на сервере и умеет взаимодействовать с созданными нами сервлетами. Иными словами, если мы хотим запустить наше веб-приложение на сервере, мы сначала разворачиваем контейнер сервлетов, а потом помещаем в него сервлеты. Схема работы проста: когда клиент обращается на сервер, контейнер обрабатывает его запрос, определяет, какой именно сервлет должен его обработать и передает его.

Обработка запроса

1. Браузер формирует http запрос и отправляет его на сервер
2. Веб контейнер создает объекты request и response
3. Веб контейнер вызывает метод service сервлета
4. Сервлет формирует ответ и записывает его в response

Жизненный цикл сервлета

1. Загрузка класса сервлета (пытаемся найти \*.java)
2. Создаем экземпляр сервлета
3. Вызываем метод init
4. Обработка http запроса
5. Вызов метода destroy

**Диспетчеризация запросов в сервлетах. Фильтры сервлетов.**

Очень часто возникает ситуация когда при запросе на один ресурс, нам надо перенаправить пользователя на другой ресурс(со страницы на страницу). В таких ситуациях нам нужно делигировать обработку запроса. Для этого нам надо получить Объект класса RequestDispatcher, с помощью которого мы сможем перемещаться на другие ресурсы нашего приложения.

* ServletRequest (абсолютный или относительный URL) – т.к. у requesta есть url.

RequestDispatcher requestDispatcher = Request.getServletDispatcher();

* ServletContext (только абсолютный URL) -

RequestDispatcher requestDispatcher = servletContext.getRequestDispatcher ()

Получив объект мы можем вызвать два метода

* Два способа делегирования обработки запроса - forward и include передав им req, resp.
  + Forward не запоминает что произошло с request response все ответственность на б - redirect
  + Include – учитывает обработку результата сервлета а – шапку сделал а, б создал тело

Фильтры

Фильтры сервлетов - это подключаемые Java-компоненты, которые мы можем использовать для перехвата и обработки запросов перед их отправкой сервлетам и ответа после завершения кода сервлета и до того, как контейнер отправит ответ обратно клиенту.

Чтобы использовать фильтр мы должны создать класс implements Filter. Переопределить методы:

* void init(FilterConfig paramFilterConfig) - Когда контейнер инициализирует фильтр, вызывается этот метод. Этот метод вызывается только один раз в жизненном цикле фильтра, и мы должны инициализировать все ресурсы в этом методе. FilterConfig используется контейнером для предоставления параметров инициализации и объекта контекста сервлета фильтру.
* doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain) Это метод, вызываемый контейнером каждый раз, когда ему нужно применить фильтр к ресурсу. Контейнер предоставляет ссылки на объекты запроса и ответа для фильтрации в качестве аргумента. FilterChain используется для вызова следующего фильтра в цепочке. Это отличный пример схемы цепочки ответственности.
* void destroy()

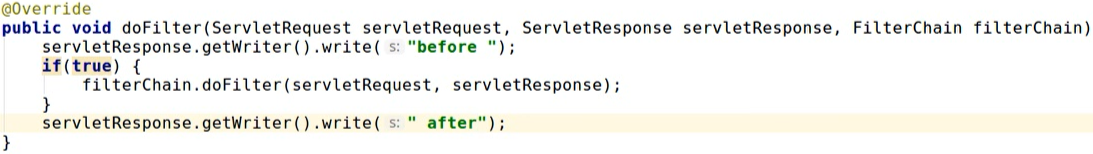
Для того чтобы связать фильтр и сервлет

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



* Фильтры позволяют осуществлять пред- и постобработку запросов до и после передачи их ресурсу (сервлету, JSP или HTML-странице).
* Пример предобработки - допуск к странице только авторизованных пользователей.
* Пример постобработки - запись в лог времени обработки запроса.
* Реализуют интерфейс javax.servlet.Filter.
* Ключевой метод - doFilter.
* Метод doFilter класса FilterChain передаёт управление следующему фильтру или целевому ресурсу; таким образом, возможна реализация последовательностей фильтров, обрабатывающих один и тот же запрос

**HTTP-сессии - назначение, взаимодействие сервлетов с сессией, способы передачи идентификатора сессии.**

Так как HTTP - stateless-протокол, то люди разработали такую вещь как сессии. По своей сути это объект, который заимплементил интерфейс HttpSession который создался на сервере, и хранит всю необходимую нам информацию для конкретного клиента какое-то время.

Экземпляр HttpSession создаётся при первом обращении клиента к приложению и сохраняется некоторое (настраиваемое) время после последнего обращения. В экземпляр HttpSession можно помещать общую для этой сессии информацию (методы getAttribute и setAttribute).

Способы передачи id сессии

* Куки – при первом обращении клиента на сервер, сервер видит что у клиента в куках(=map) нет ключа значение id сессии, тогда он создает сессию, и передает клиенту в куках его id, и в будущем у клиента в куках уже есть значение сессии и при запросе сервер получает его и работает с его объектом sessison. Куки пересылаются вместе с каждым запросом через заголовоки:
  + Set-Cookie: имя = значение - Сервер отправляет следующее в своем заголовке ответа, чтобы установить поле cookie.
  + Cookie: имя = значение - Если есть набор файлов cookie, то браузер отправляет следующее в заголовке запроса.
* Через URL - session\_name=session\_id - id начнет передаваться в url автоматически браузером.
* Параметр GET или POST запросов HTTP

Пример работы с сессией

HttpSession session = request.getSession(); - получение сессии

session.setAttribute("name", "Tom Soyer"); - создать пару ключ - значение

String name = (String) session.getAttribute("name"); - получить значение

session.removeAttribute("name"); - удалить ключ значение

**Контекст сервлета - назначение, способы взаимодействия сервлетов с контекстом.**

* API, с помощью которого сервлет может взаимодействовать со своим контейнером.
* Доступ к методам осуществляется через интерфейс javax.servlet.ServletContext.
* У всех сервлетов внутри приложения общий контекст.
* В контекст можно помещать общую для всех сервлетов информацию (методы getAttribute и setAttribute).
* Если приложение - распределённое, то на каждом экземпляре JVM контейнером создаётся свой контекст.

ServletContext selvletContext = getServletContext();

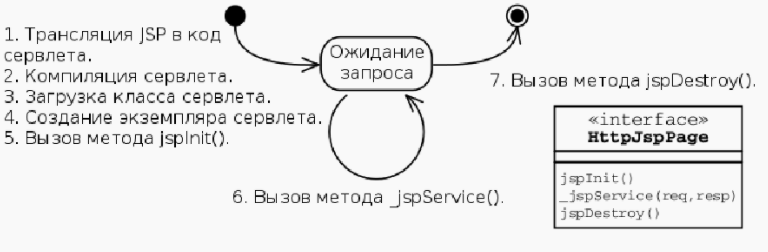
selvletContext.setAttribute("name", "Tom");

selvletContext.setAttribute("age", 35);

**JavaServer Pages. Особенности, преимущества и недостатки по сравнению с сервлетами, область применения.**

* Страницы JSP - это текстовые файлы, содержащие статический HTML и JSP-элементы.
* JSP-элементы позволяют формировать динамическое содержимое.
* При загрузке в веб-контейнер страницы JSP транслируются компилятором (jasper) в сервлеты.
* Позволяют отделить бизнес-логику от уровня представления (если их комбинировать с сервлетами).
* Преимущества:
  + Высокая производительность - транслируются в сервлеты.
  + Не зависят от используемой платформы – код пишется на Java.
  + Позволяют использовать Java API.
  + Простые для понимания - структура похожа на обычный HTML.
* Недостатки:
  + Трудно отлаживать, если приложение целиком основано на JSP.
  + Возможны конфликты при параллельной обработке нескольких запросов.

**Жизненный цикл JSP.**



**Структура JSP-страницы. Комментарии, директивы, объявления, скриптлеты и выражения.**

JSP может быть написан на html или xml может включать элементы

* Комментарии <%-- … --%> - jsp ; <!-- … --!> html ; <% //… %> - java
* Директивы – команды для джаспера т.е. они как то влияют на трансляцию jsp страницы include
* Объявления – <%! … %> фрагмент джава кода, который будет с транслирован в тело класса. Поля методы
* Скриплеты – jsp (на джава) элементы, которые в дальнейшем будут cтранслированы в тело метода service.
* Выражения – конструкция, которая при трансляции в сервлет будет странслирована в out println

**Правила записи Java-кода внутри JSP. Стандартные переменные, доступные в скриптлетах и выражениях.**

JSP неявные объекты JSP контейнер предусмотрен для каждой страницы объектов Java, разработчики могут использовать их непосредственно, без явного объявления. JSP неявные объекты также известны как предопределенные переменные.

* **Request**: объект запроса является экземпляром класса. Объект запроса предоставляет ряд методов, чтобы получить HTTP-заголовки, куки, методы HTTP, и так далее.
* **Response**: Объекты ответа также определены в интерфейсе модуля обработки заголовка HTTP. С помощью этого объекта, разработчики могут добавлять новые куки, метку времени, HTTP код статуса, и так далее.
* **session**: объект сеанса используется для отслеживания сеанса между каждого запроса клиента
* **application**: Этот объект на протяжении всего жизненного цикла страницы JSP представляет собой страницу JSP. Этот объект создается, когда страница JSP инициализируется с помощью вызова метода jspDestroy () был удален. Путем добавления атрибутов к приложению, а затем все компоненты вашего веб-приложения JSP-файлы могут обращаться к этим свойствам
* **config**: Этот объект позволяет разработчикам получить доступ к параметрам инициализации сервлета или двигатель JSP, такие как пути к файлам.
* **pageContext**: Этот объект используется для доступа к основной информации страницы, отфильтровывая большинство деталей реализации.
* **Page**: Объект страница является синонимом для этого объекта.
* **Exception**: Он часто используется для создания соответствующего ответа на состояние ошибки.

**Bean-компоненты и их использование в JSP.**

Компоненты JavaBeans представляют собой классы Java, которые могут быть легко использованы повторно и объединены в приложение. Любой класс Java, который соответствует определенным требованиям проектирования, может быть компонентом JavaBeans.

Требования

* Get/set
* Конструктор без параметров
* Serializiable

Изображение выглядит как текст

Автоматически созданное описаниеРабота с Bean-ами в JSP происходит через JSP Actions:

Пример использования:

<jsp:useBean id = "name" class = "class" scope = "session">

<jsp:setProperty name = "name" property = "property1" value = "value"/>

<jsp:getProperty name = "name" property = "property2 "/>

</jsp:useBean>

**Стандартные теги JSP. Использование Expression Language (EL) в JSP.**

Чтобы упростить встраивание кода java в JSP была разработана специальная библиотека - JSTL. JSTL (JSP Standard Tag Library) предоставляет теги для базовых задач JSP (цикл, условные выражения и т.д.). Эта библиотека не является частью инфраструктуры Java EE, поэтому ее необходимо добавлять в проект самостоятельно.

JSTL является альтернативой скриплетам, встроенным в JSP, то есть прямым вставкам Java кода. Представляет собой набор тегов в стиле HTML, позволяющих обращаться к объектам Java и выполнять многие из конструкций языка Java.

Несмотря на то, что JSTL часто называется библиотекой, на самом деле она содержит ряд библиотек:

* **Core**: содержит основные теги для наиболее распространенных задач. Использует префикс "c"
  + <c:out> используется для вывода данных на экран
  + c:set используется для определения переменной
* **Formatting**: предоставляет теги для форматирования чисел, дат, времени. Использует префикс "fmt"
* **SQL**: предоставляет теги для sql-запросами и источниками данных. Использует префикс "sql"
* **XML**: предоставляет теги для работы с xml. Использует префикс "x"
* **Functions**: предоставляет функции для работы со строками. Использует префикс "fn"

Для подключения функционала этих библиотек на страницу jsp применяется директива taglib.

Изображение выглядит как текст

Автоматически созданное описаниеПример использования

Expression Language (EL) в JSP

Expression Language или сокращенно EL предоставляет компактный синтаксис для обращения к массивам, коллекциям, объектам и их свойствам внутри страницы jsp. Он довольн прост. Вставку окрывает знак $, затем в фигурные скобки {} заключается выводимое значение

Откуда эти данные берутся? EL пытается найти значения для этих данных во всех доступных контекстах. И EL просматривает все эти контексты в следующем порядке:

* Контекст страницы (данные сохраняются в PageContext
* Контекст запроса
* Контекст сессии
* Контекст приложения

Примеры <p>Name: ${requestScope.name}</p> <p>Name: ${name}</p>

**Параметры конфигурации JSP в дескрипторе развёртывания веб-приложения.**

В общем случае jsp конфигурировать не надо, он сам будет работать на уровне веб контейнера. Однако, если мы захотим, то мы сможем сами задать её в десрипторе развертывания web.xml, будет находиться внутри элемента jsp-config

**Шаблоны проектирования и архитектурные шаблоны. Использование в веб-приложениях.**

Шаблон проектирования или паттерн - повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста (© Wikipedia). Описывает подход к решению типовой задачи. Одну и ту же задачу часто можно решить с использованием разных шаблонов.

Порождающие – для создания объектов

* Abstract Factory - Абстрактная фабрика.
* Builder - Строитель.
* Factory Method - Фабричный метод.
* Prototype - Прототип.
* Singleton - Одиночка.

Структурные - обычно решают задачу совместного использования классов, многие из них являются частными случаями делегирования

* Adapter - Адаптер.
* Bridge - Мост.
* Composite - Компоновщик.
* Decorator - Декоратор.
* Facade - Фасад.
* Flyweight - Приспособленец.
* Proxy - Заместитель.

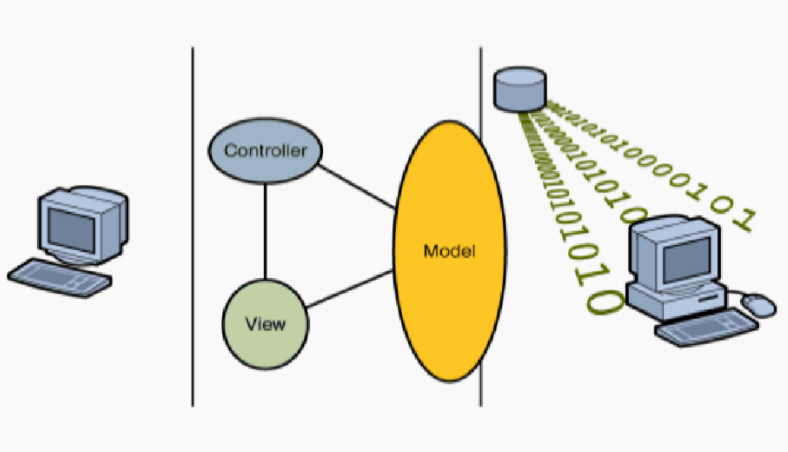
Поведенческие - реализуют различные варианты поведения. Обычно они позволяют динамически управлять ответственностью.

* Chain of responsibility - Цепочка обязанностей.
* Command - Команда.
* Interpreter - Интерпретатор.
* Iterator - Итератор.
* Mediator - Посредник.
* Memento - Хранитель.
* Observer - Наблюдатель.
* State - Состояние.

Архитектурные шаблоны

* Более высокий уровень по сравнению с шаблонами проектирования.
* Описывают архитектуру всей системы или приложения.
* Обычно имеют дело не с отдельными классами, а с целыми компонентами или модулями.
* Компоненты и модули могут быть построены с использованием различных шаблонов проектирования.

**Архитектура веб-приложений. Шаблон MVC. Архитектурные модели Model 1 и Model 2 и их реализация на платформе Java EE.**



View – реализует уровень представления данных пользователю jsp

Model – данные

Conroller – куда пришел запрос и что делать